Una Metodología de Educación Basada en Componentes

Sergio Ochoa, David Fuller
Pontificia Universidad Católica de Chile
Escuela de Ingeniería. DCC
Santiago, Chile
{sochoa, dfuller}@ing.puc.cl

Resumen

El presente trabajo aporta una propuesta tanto metodológica como tecnológica, para mejorar la calidad de la educación, independizándose en alguna medida, del docente que la imparte. Para ello se propone una metodología de desarrollo de cursos, basada en tecnología, que apoya la labor del profesor, de los auxiliares, y de los alumnos. Observando las similitudes de los problemas de la educación actual y la de la ingeniería de software en la época de la crisis del software, se presenta una solución que transporta al área de educación, las soluciones más generales que ayudaron a la ingeniería de software a salir de la crisis. En un principio la solución está orientada a educación universitaria, pero es fácilmente extendible a la educación media y básica.

Palabras Claves: Componentes Educativos, Educación Centrada en el Alumno, Patrones de Diseño.

1. Introducción

El presente trabajo se enmarca en la intersección entre la ingeniería de software y la educación. En los últimos años, los investigadores tanto de áreas tecnológicas, como de las educativas, han trabajado intensamente con psicólogos, sociologos, diseñadores gráficos, etc., en el desarrollo de estratégias que ayuden a mejorar la calidad de la educación, apoyándose en las capacidades que hoy brinda la tecnología. Estos trabajos han dado frutos, que han sido reflejados en muchas de las herramientas conocidas como WCB (Web Course in a Box). Aunque la orientación actual de estas herramientas, es hacia la educación a distancia, y no hacia la educación efectiva.

Hablar de mejora en la calidad de la educación, es hablar en términos muy generales, ya que hay demasiados aspectos que pueden ser impactados, y que producirían una mejora real. En nuestro caso, el problema que hemos atacado, es la "*naturaleza artesanal del proceso de impartir conocimientos*". En la actualidad, la calidad de la educación entregada en un curso depende casi en su totalidad del profesor a cargo. Y si bien es imposible independizarse de las habilidades del profesor, los trabajos realizados en otras áreas indican que es posible garantizar una calidad de educación mínima (base). Específicamente, nuestra propuesta consiste en:

> *"Transformar el **proceso artesanal** de enseñanza-aprendizaje, en un **proceso evolutivo y profesional**, orientado a **garantizar una calidad de educación mínima**, en forma independiente del instructor".*

Este problema no es nuevo, de hecho la ingeniería de software ha tenido que abordarlo, para poder superar la etapa de "*la crisis del software*". En el presente trabajo se presenta una analogía, tanto del problema como de la solución, como justificación de la propuesta.

1.1. Características de sistemas educativos actuales

La entrega de conocimientos a través de un curso, está organizada como un proceso de naturaleza "Artesanal", por eso caímos en la actual Crisis Educativa **[Harvey 98]**.

Problemas:

- En la velocidad y efectividad de adquisición de conocimientos.
- Procesos artesanales
- Niveles Bajos de Comunicación (métodos expositivos)
- Contenidos sin estructura (vínculos con el conocimiento almacenado)
- Control de calidad pobre (evaluaciones):

Diagnóstico: No se puede garantizar el resultado final, éste depende fundamentalmente del profesor.

1.2. Características de la ingeniería de software, durante la "crisis del software".

La Ingeniería de Software estaba organizada como un proceso de naturaleza "Artesanal", por eso caímos en la Crisis del Software.

Crisis del Software:

- Procesos artesanales
- Bajos niveles de comunicación
- No hay una estructura clara a seguir (¿qué pasos se deben seguir? y ¿qué documentos se deben producir?, ¿Con qué estructura?)
- Hay muy poco testing, y además no está automatizado.

Diagnóstico: No se puede garantizar el resultado final, éste depende fundamentalmente de los integrantes del equipo de trabajo.

1.3. Evolución de la Ingeniería de Software

La primera, y una de las mejores soluciones, fue propuesta por Barry Boehm en 1988, la cual fue popularizada como el *modelo espiral* **[Boehm 88].** Esta solución plantea la necesidad de iterar permanentemente sobre el tradicional ciclo de desarrollo de software. Partiendo desde los requisitos, hasta la implementación y entrega del producto. Cada ciclo comienza con una comunicación con el usuario (etapa 1), para establecer claramente el objetivo a alcanzar durante la iteración, y los requisitos asociados a él. A partir de objetivos claros, es posible planificar el trabajo a desarrollar (etapa 2). Una vez confeccionada la planificación, se analizan los riesgos a los que estará sometido el proyecto (etapa 3), durante esa iteración, y se planifican acciones correctivas y/o proactivas. Luego se pasa a la etapa de ingeniería, donde se diseña en forma detallada lo que se desea construir (etapa 4), e inmediatamente a continuación se construye (etapa 5). Por último, los resultados obtenidos se evalúan con el usuario (etapa 6), y dependiendo de su nivel de conformidad, se comienza o no una nueva iteración, redefiniendo los objetivos a alcanzar durante el nuevo ciclo. Esta metodología plantea un enfoque "top-down" para el desarrollo de sistemas, ya que a partir del diseño arquitectónico, cada iteración va agregando detalles, hasta llegar a la implementación. El enfoque "top-down" y evolutivo, ha hecho que el riesgo de apartarse de los objetivos especificados, se reduzca notablemente, debido a la reducción del tamaño de los problemas a atacar (y por lo tanto la complejidad) durante una iteración.

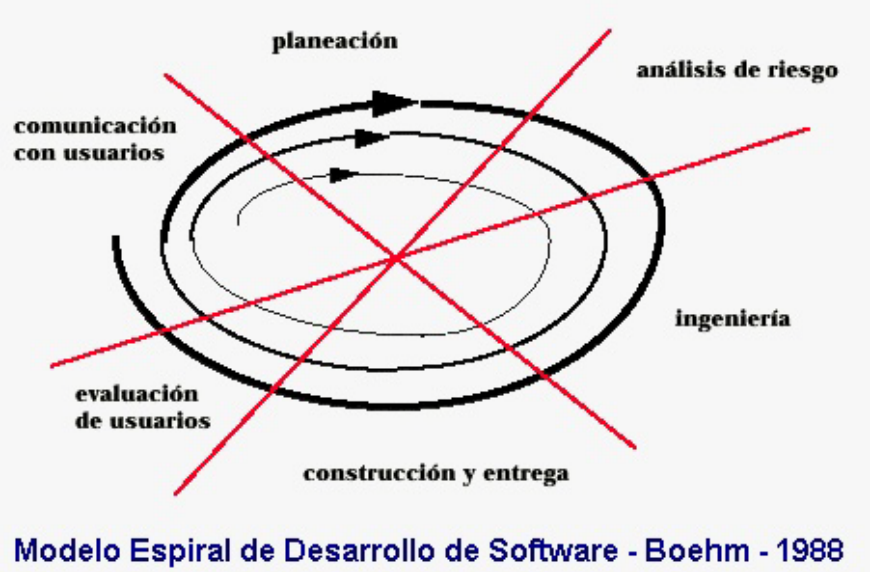

Figura 1: Modelo Espiral

Como una extensión del modelo espiral, Nierstrasz planteó el primer modelo basado en componentes **[Nierstrasz 92]**, que consistía en desarrollar software usando el modelo de Bohem y una gran librería de clases. Obviamente este enfoque apuntaba directamente a la reutilización de código.

El modelo de Nierstrasz, como se puede ver en la *figura 2*, reemplaza las etapas de ingeniería, construcción y entrega, por una única etapa de *construcción y adaptación de la ingeniería*. El objetivo de ésta es desarrollar el sistema basándose en los componentes de software almacenados en librerías. Para ello se identifican a partir del diseño, componentes necesarios para la implementación, luego se buscan y se extraen de la biblioteca y por último se integran a la solución. En el caso de que la búsqueda resulte infructuosa, se construyen el o los componentes necesarios y se publican en la librería de clases. La idea es construir sólo si no existe (ni se puede adaptar) ningún componente capaz de desarrollar la tarea en cuestión. Los componentes construidos pueden utilizar a otros componentes existentes en la librería, y por lo tanto pueden ser tan complejos como se desee.

Debido a la utilización de este modelo, QSM Associates, Inc. reportó en 1994 **[Pressman 97]**, reducciones del orden de:

70 % en el Tiempo de Desarrollo
84 % del Costo de Desarrollo

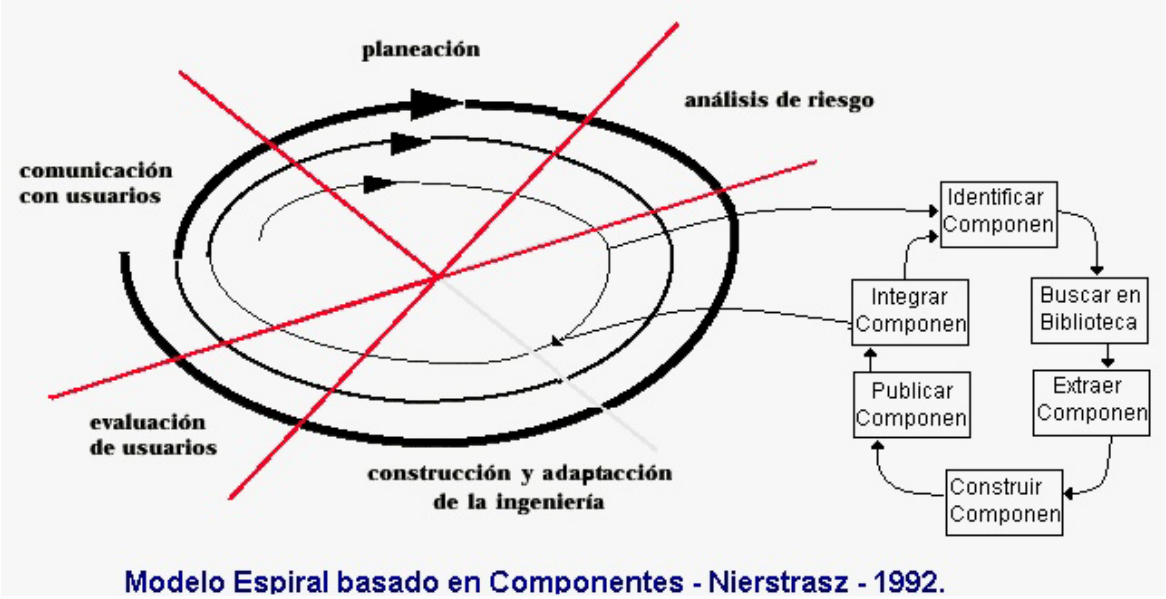

Figura 2: Modelo de Nierstrasz

Christopher Alexander en 1975 introdujo un nuevo concepto en diseño, llamado el "*Patrón de Diseño*" **[Alexander 77]**, y lo define como "*cada patrón describe un problema que ocurre una y otra vez en nuestro ambiente, y luego describe la esencia de la solución al problema, de manera que se pueda utilizar esa solución un millón de veces, y nunca hacerlo dos veces del mismo modo*". Aunque este concepto revolucionario hoy se aplica en muchas áreas, la demora en transportar el concepto al software fue de 20 años. Erich Gamma y sus colaboradores, fueron los encargados de establecer los primeros patrones de software, y los definieron como "*un patrón nombra abstrae e identifica los aspectos claves de una estructura de diseño común, que lo hace útil para crear un diseño reutilizable*" **[Gamma 95]**. Este fue el siguiente paso en la evolución de las metodologías de desarrollo de software. Luego, la evolución en esta área llevó a la definición de componentes en el sentido actual, o sea *componentes de software que soportan un conjunto de comportamientos estándares, independientemente de la funcionalidad específica para la que han sido diseñados* **[Vander Veer 97]**. Fue entonces cuando la librería de clases del modelo de Nierstrasz, se convirtió en una librería de componentes.

No todos los componentes pueden definirse como patrones, aunque los patrones sí pueden ser manejados como componentes. Muchos de los componentes actuales reflejan patrones **[Booch 98, Brown 97, Meyer 99]**, o lo que es lo mismo, portan el conocimiento respecto a algo (conocimiento del patrón). Las dos ventajas más importantes del uso de componentes radican en la reusabilidad, y en la capacidad de reflejar conocimiento adquirido durante años, por investigadores expertos en distintas áreas. Actualmente, las librerías de componentes son organizadas en un "framework", que es capaz de agrupar a los componentes relacionados, y de agregarle semántica a estas relaciones. Roberts y Johnson definen a un framework como "*diseños reusables de partes o de todo un sistema de software, descripto por un conjunto de clases abstractas, y por la manera en que las instancias de ellas colaboran*" **[Roberts 97]**.

1.4. Herramienta de Medición

Esta evolución de la ingeniería de software, ha permitido que la capacidad de desarrollo de software de una empresa, sea más previsible, e independiente de su personal. Además de mejorar notablemente la calidad del producto final. Cuanto mayor es el nivel de independencia de la empresa, con respecto de sus empleados, se dice que la empresa es más "madura". Todas las metodologías que hemos revisado antes, apuntan a aumentar la madurez de la empresa desarrolladora, de esa manera se volverá mas predecible.

Para poder medir el grado de madurez de una organización, se utiliza generalmente la propuesta del SEI (Software Engineering Institute), conocida como CMM (Capability Maturity Model) **[Paulk 91].** Esta propuesta, categoriza a las organizaciones sobre una escala de 1 a 5 según el grado de madurez alcanzado.

Como resumen de todo lo anterior podemos decir que los problemas que tenía la ingeniería de software durante "la crisis del software" **[Pressman 97, Sommerville 95]**, fueron en gran medida solucionados con un modelo evolutivo e iterativo, con el agregado de librerías de componentes. Estos resultados han sido medidos utilizando CMM, y según Roger Pressman, "las mejoras de productividad se encuentran en el intervalo que media entre el 25 al 40 %" **[Pressman 97],** por el sólo hecho de usar componentes**.** Es por eso que en el presente trabajo, se transportan al área de educación, las soluciones alcanzadas por la ingeniería de software, con el objetivo de ayudar a "madurar" a las instituciones educativas.

2. Estado del Arte

Existe una gran cantidad de herramientas tecnológicas basadas en WEB, que permiten implementar cursos. La mayoría de estos ambientes están en una etapa de constante evolución, y si bien es cierto se están utilizando en algunas universidades, se consideran aún un estado experimental. Este tipo de herramientas son comúnmente conocidas como WCB (Web Course in a Box) **[Landon 98]**, y las más distinguidas son:

- **Lotus Learning Space** de la empresa IBM Lotus Corp. **[LerningSpace 99]**
- **WebCT** desarrollado en el Departamento de Ciencia de la Computación de la University of British Columbia, Canadá **[WebCT 99]**
- **TopClass** de la empresa WBT Systems (anteriormente llamada WEST) **[TopClass 98]**
- **Virtual-U** desarrollado por la Simon Fraser University, British Columbia, Canadá **[Virtual-U 99]**
- **WCB** de Virginia Commonwealth University **[Wcb 98]**
- **CourseInfo** de la empresa Blackboard. Inc. **[CourseInfo 99]**
- **ToolBook II**, es una familia de productos comerciales de Asymetrix. Learning System Inc. **[ToolBook 99]**

Algunas de las ventajas más importantes que éstas aportan son:

- Automatizan de los cursos en un gran porcentaje.
- Mantienen la memoria histórica del curso.
- Mejora la calidad del material de apoyo.
- Aumentan de la comunicación y colaboración entre los participantes.
- Rompen con las barreras de tiempo y distancia.

Independientemente de los aspectos psicológicos, sociales, culturales, idiomáticos, etc., estas herramientas tienen a su vez otras desventajas asociadas:

- Los productos no tienen capacidad de ajuste a situaciones particulares
- No existe un modelo instruccional para la implementación de los cursos.
- Es difícil poder reutilizar el conocimiento, en otros cursos.
- El apoyo que le brindan al docente, es su tarea de transmitir el conocimiento, es muy pobre.

Con respecto al uso de componentes de software en el área de educación, no hay antecedentes. Tampoco hay reportes de experiencias realizadas para transportar, al área de educación, las soluciones alcanzadas por la ingeniería de software.

3. Solución Propuesta

Debido a que las características de los problemas de la Ingeniería de Software y de la Educación son similares, la solución propuesta también es similar. Específicamente se plantea la definición de:

- Un Modelo guía, evolutivo y genérico, para la implementación de cursos.
- Framework de componentes, orientado a la educación.

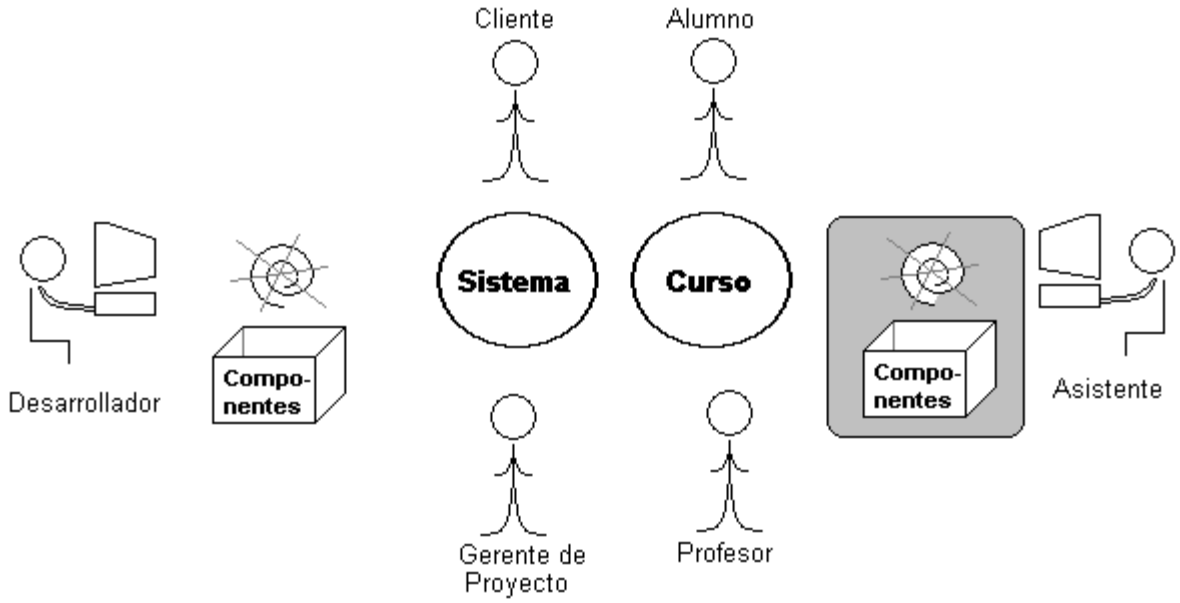

Figura 3: Analogía entre Desarrollo de Software y Desarrollo de Cursos

La figura 3 muestra gráficamente la analogía que ya se ha presentado, mostrando también las similitudes en los roles de los participantes. A continuación se describe el modelo educativo propuesto.

3.3. Descripción del Modelo Educativo

Al igual que el modelo de Nierstrasz, se comienza con la definición de los objetivos, y de los contenidos más adecuados para alcanzarlo (Etapa 1). Luego, teniendo en claro los objetivos y los contenidos, es necesario diseñar la mejor forma de presentarlos (Etapa 2). Esto se refiere tanto a la selección del material a utilizar, como al método instruccional elegido para presentación del conocimiento. A continuación se realiza un análisis de riesgo, fundamentalmente cuando se utilizan tecnologías inmaduras o métodos poco probados (Etapa 3). Luego se procede a la implementación (Etapa 4), que puede ser la construcción de alguna parte del material del curso, o una actividad grupal (como por ejemplo una lluvia de ideas), o la presentación de algún tema, etc. Con respecto a la utilización de la librería de componentes, se sigue la misma idea expuesta en el punto 1.3, teniendo en cuenta que un componente puede ser tan simple como un ejemplo o tan complejo como un foro de discusión estructurado o un ambiente educativo. Finalmente, es necesario obtener retroalimentación para observar el nivel de satisfacción de los alumnos (Etapa 5). En caso de ser necesario, el objetivo puede ser rediseñado, y en ese caso se comenzaría con un nuevo ciclo.

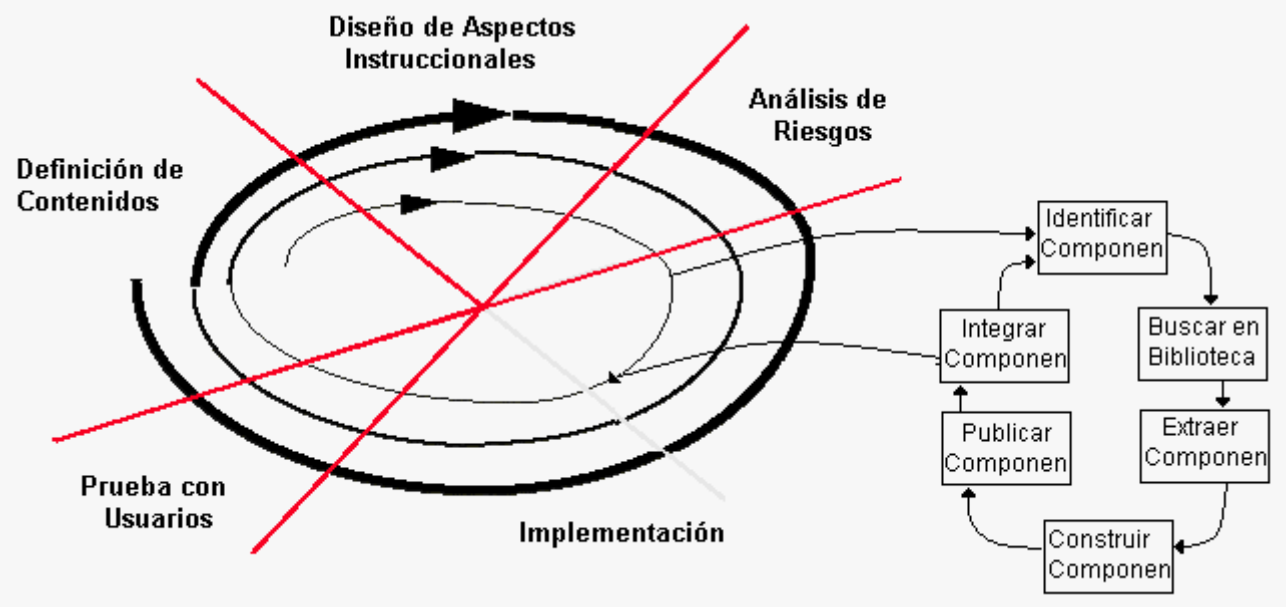

Figura 4: Modelo Propuesto para el Desarrollo de Cursos

3.4. Framework Colaborativo Orientado a la Educación

Para la construcción de los componentes, es fundamental el trabajo interdisciplinario de ingenieros, psicólogos, sociólogos, educadores, diseñadores gráficos, etc., principalmente en todo lo relativo a la interfaz con el usuario **[Mullet 95, Schneiderman 98]**. Es sabido que el trabajo interdisciplinario es difícil de llevar a cabo, por todas las diferencias de lenguaje, costumbres y madurez de la ciencia involucrada en cada área. Y también que un maestro, como cualquier ser humano, es imposible que esté familiarizado con todos los aspectos que ejercen alguna influencia sobre su trabajo. Y aunque lo estuviera, es muy difícil que pueda preparar sus clases como él lo que desea, debido al poco tiempo que posee para esa actividad. Es por ello que proponemos *"una solución basada en componentes"*, que tiene la capacidad de sugerir (comportamiento por defecto), y además tiene la capacidad de reutilizar tanto el trabajo propio como el ajeno.

"Los componentes tienen la capacidad de poder transportar el conocimiento almacenado en él" **[Crane 97].** Existen distintos tipos de componentes, aunque todos mantienen la misma filosofía. En nuestro caso en particular, estamos hablando de JavaBeans distribuidos a través de RMI (Remote Method Invocation). Según Vander Veer **[Vander Veer 97]** "*los JavaBeans, son componentes de software funcionalmente discretos, con los cuales se pueden construir complejas aplicaciones basadas en java*".

Imaginemos por un momento que deseamos construir un componente llamado *programa*, cuya finalidad es apoyar al profesor en la enseñanza de algoritmos de programación. ¿Cuál o cuáles son los modelos instruccionales más adecuados para desarrollar esta tarea? ¿Cómo aplicar estos modelos?. Estas son preguntas obvias para aquel que alguna vez tuvo que realizar esta tarea. La solución no tiene mucho que ver con la informática, más bien tiene que ver con el diseño instruccional. Lo bueno de los componentes es que, en su estructura, puede quedar reflejado el o los modelos instruccionales más adecuados, según lo que establezcan los expertos en el tema. Y todo este conocimiento y trabajo interdisciplinario almacenado en el componente, es reutilizable en un 100 %. Parte del framework de componentes propuesto, se muestra en la figura 5.

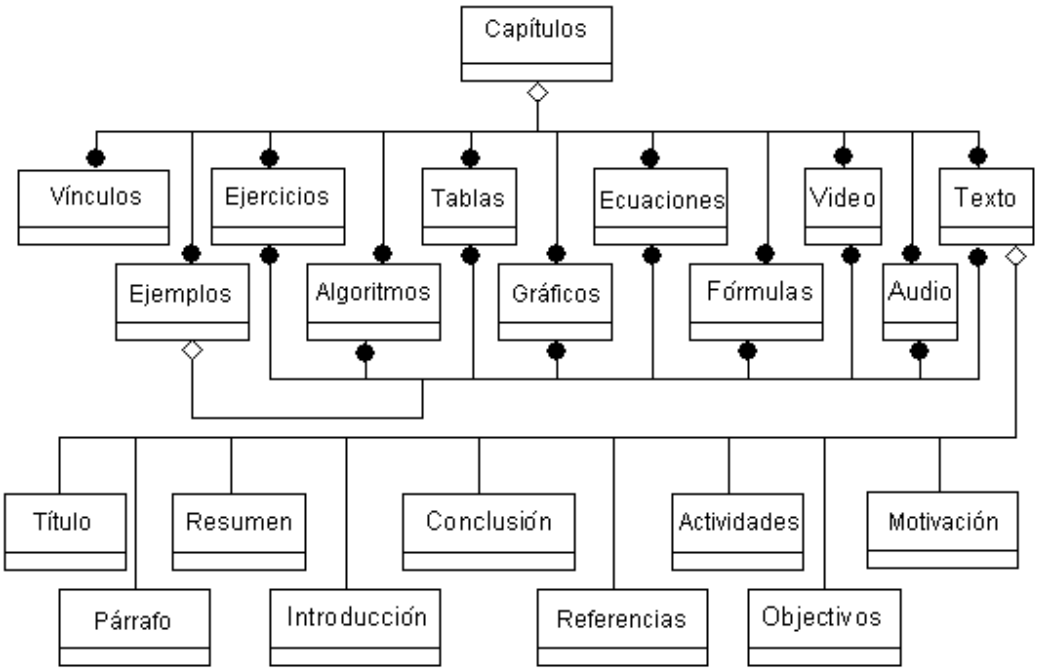

Figura 5: Parte del Framework de Componentes Propuesto

En la actualidad existe un mercado emergente alrededor de los componentes. Muchas empresas desarrollan y comercializan componentes tanto genéricos como específicos, para algún área en particular. Hasta este momento, no hemos encontrado reportes con respecto al desarrollo de componentes en el área de educación.

5. Trabajo Futuro

Con respecto al trabajo futuro podemos decir que en este momento la metodología está en la etapa final de su diseño, en cambio el framework está en la etapa inicial. A futuro se deben terminar estas tareas, y luego realizar varios experimentos para tratar de medir alcances y limitaciones de la propuesta.

Sería muy interesante que se trabajara en una versión similar al CMM, pero para certificar el nivel de madurez de las Instituciones Educacionales, basado en los recursos que utilizan para enseñar (Tecnología, Métodos Instruccionales, Materiales Educativos, etc.). Una guía como esta, le ayudaría a muchas instituciones mejorar en forma continua la calidad de la enseñanza que están ofreciendo. Hasta ha habido avances en algunas líneas específicas, como por ejemplo, la certificación ABET (Accreditation Board for Engineering and Technology) **[Abet 98]**.

6. Conclusiones

Es importante investigar las similitudes (patrones) en los problemas que tenemos que enfrentar, especialmente si éstos son complejos. De hecho Martín Fowler en su libro “Patrones de Análisis” **[Fowler 97]**, hace una analogía entre el funcionamiento de una bicicletería y de un Hospital, y demuestra que ambos siguen el mismo “patrón de análisis”, por lo tanto la experiencia en un área puede ser transportable a la otra.

Este trabajo propone un enfoque inédito en el área de educación asistida por computador, y si bien los resultados no pueden conocerse sino hasta realizar un experimento científicamente diseñado, las ganancias obtenidas por la ingeniería de software hacen esperar un aporte real, de nuestra propuesta. Si los componentes de software diseñados logran el impacto planificado, es probable que comience a generarse un mercado de comercialización de conocimiento basado en componentes, donde habrá

oferta y demanda de módulos de conocimiento, específicos para enseñar un tema en particular. Dichos componentes reutilizarán la estructura (JavaBeans), aunque el conocimiento almacenado y el método de enseñanza será específico para cada componente. Además, debido a que se ajustan al estándar especificado por Sun Microsystems, tienen la capacidad de comunicarse con otros componentes, con la misma o con distinta estructura.

7. Agradecimientos

Este trabajo ha sido parcialmente financiado por el Fondo Nacional de Ciencia y Tecnología de Chile (FONDECYT), Nº 198-0960.

Queremos agradecer al Profesor Luis Alberto Guerrero, del Departamento de Ciencia de la Computación, de la Universidad de Chile, por sus contribuciones al presente trabajo.

8. Referencias

[Abet 98] Accreditation Board for Engineering and Technology, Inc. Internet Homepage. 1998. http://www.abet.org/

[Alexander 77] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I.Fiksdahl-King, S. Angel. "*A Pattern Language*". Oxford University Press, New York. 1977.

[Boehm 88] B. Boehm. "*A Spiral Model for Software Development and Enhacement*". Computer, Vol.21, nº 5, May. 1988.

[Booch 98] G.Booch, W. Kozaczynski. *Component-Based Software Engineering*. IEEE Software, Vol.15, Nro.5, pp34-36, Sep/Oct. 1998.

[Brown 97] A. Brown, K. Short. *On Components and Objects: Foundations of Component-Based Development.* Proc. of the 5th International Symposium on Assesment of Software Tools. IEEE Computer Society Press. Jun. 1997.

[CourseInfo 99] Blackboard, Inc. "*CourseInfo*". 1999. http://product.blackboard.net/courseinfo/

[Crane 97] A. Crane, S. Clyde. "*Extending Patterns for GUI Design*". Research Paper. Utah State University. 1997.

[Fowler 97] M. Fowler. "*Analysis Patterns*". Addison-Wesley. 1997.

[Gamma 95] E. Gamma, R. Helm, R. Johnson, J. Vissides. "*Design Pattern: Elements of Reusable Object-Oriented Software*", Addison Wesley. 1995.

[Harvey 98] L. Harvey (1998), "*La educación en el siglo XXI*", Calidad en la Educación Superior, Consejo Superior de Educación, Chile

[Landon 98] B. Landon. "*OnLine educational delivery applications: a web tool for comparative analysis*". Standing Commitee ON Educational Technology; Centre for Curriculum, Transfer and Technology: Office of Learning Technologies. 1998. http://www.ctt.bc.ca/landonline/

[LearningSpace 99] IBM Lotus Corporation. "*LearningSpace*". 1999. http://www.lotus.com/home.nsf/tabs/learnspace

[Nierstrasz 92] Nierstrasz. "*Component-Oriented Software Development*". CACM, vol.35, nº 9, Sept. 1992.

[Meyer 99] Meyer, B. 1999. *On to Components*. IEEE Computer, Jan. 1999.

[Mullet 95] K. Mullet & D. Sano. "*Designing Visual Interfaces: Communication Oriented Techniques*", SunSoft Press (Prentice-Hall). 1995.

[Paulk 91] M. Paulk, B. Curtis, M.Chrissis, C. Weber. "*The Capability Maturity Model For Software, Version 1.1.*". Software Engineering Institute, Carnegie Mellon University. 1991. http://www2.umassd.edu/SWPI/SEI/tr24f/tr24.html

[Pressman 97] R. Pressman "*Ingeniería del Software, un Enfoque Práctico*", 4º Ed., Mc. Graw Hill. 1997.

[Roberts 97] D. Roberts, R. Johnson. "*Patterns for Evolving Frameworks*", PLOP-3 (Pattern Language Oriented Programing). 1997.

[Shneiderman 98] B. Shneiderman. *Designing User Interface: Estrategies for Effective Human Computer Interaction*, 3º Ed. Addison-Wesley Publising Company. 1998.

[Sommerville 95] I. Somerville. Software Engineering. 5th Edition. Addison Wesley. 1995.

[ToolBook 99] Asymetrix Learning System, Inc. "*ToolBook II*". 1999. http://www.asymetrix.com/
[TopClass 98] WBT Systems. "*TopClass*". 1998. http://www.wbtsystems.com/
[Vander Veer 97] E.Vander Veer . "*Beans Basics*". Web Techniques Magazine, Vol. 2. Oct.1997.
[Virtual-U 99] Simon Fraser University, British Columbia, Canadá. "*Virtual-U*". 1999. http://virtual-u.cs.sfu.ca/vuweb/
[Wcb 98] Virginia Commonwealth University, U.S.A. "*WCB (Web Course in a Box)*". 1998. http://www.madduck.com/wcbinfo/wcb.html
[WebCT 99] University of British Columbia, Canadá. "*WebCT*". 1999. http://homebrew1.cs.ubc.ca/webct/